FUNDAMENTOS DE INFORMÁTICA

TRABAJO PRÁCTICO FINAL

Profesora: WENDDY SEGOVIA

Alumno: MARTÍN ARIEL PAEZ

34137460

COMISIÓN 5

UNAJ

28/11/2021

Introducción

Las hojas de cálculo de Google Sheet son un recurso valioso para cualquier emprendedor, sin embargo, en muchas ocasiones resultan insuficientes o muy permisivas, o sea, que el usuario común podría llegar a modificar su contenido hasta el punto de afectar su funcionamiento habitual. Es por ello que, para el trabajo final de Fundamentes de informática, se decidió dar una respuesta a este inconveniente través de una aplicación escrita en Python.

El objetivo era obtener un producto aplicable a cualquier proyecto que administre su información a través de este tipo de archivos, de modo tal que dos emprendimientos diferentes puedan hacer uso de sus servicios por igual, y, sin necesidad de volver a insertar todos los datos a mano.

Ejemplo de aplicación

El archivo ejemplo_de_aplicacion.py es una implementación que hace uso del software para aplicarlo a un instituto privado. En esta sección se muestra el resultado obtenido y se irá explicando a lo largo del informe su implementación.

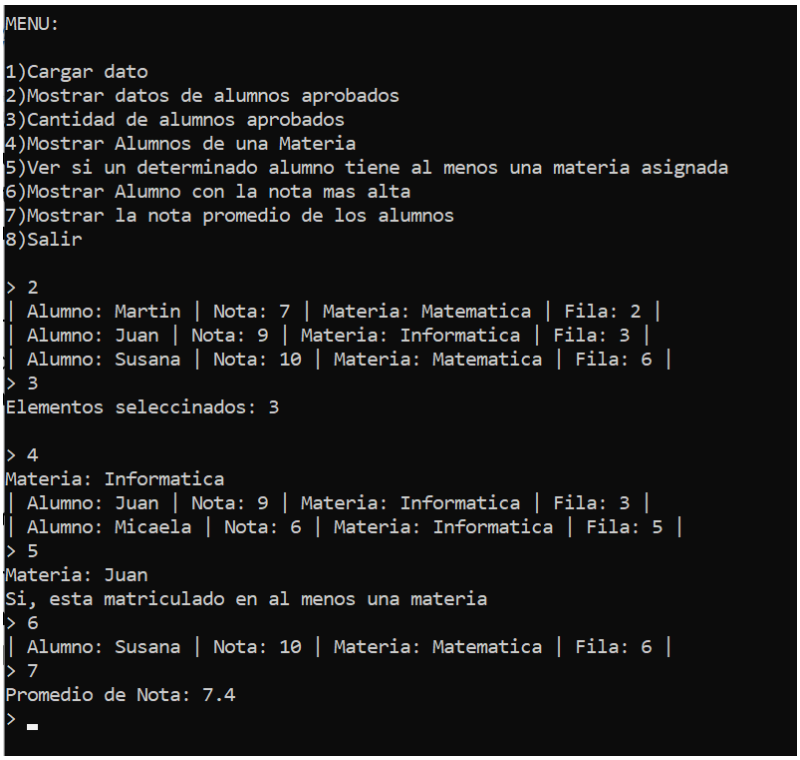

Imagen 1. Aplicado a un Instituto privado

Manual de Usuario

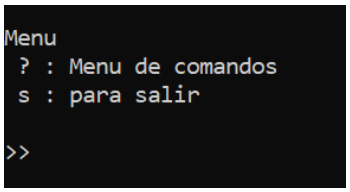

Imagen 2. Inicio del programa

Al ejecutar el programa nos encontramos con una línea de comandos. Como indica en pantalla, podemos acceder a un menú ingresando "?". Este primer menú es informativo, no pretende dar a elegir al usuario entre una serie de funciones.

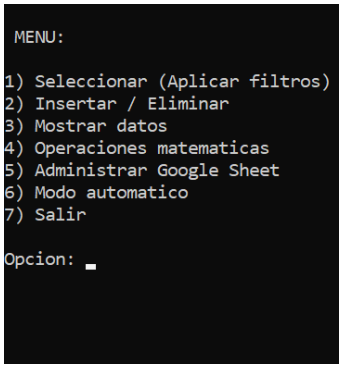

Imagen 3a. El menú "?" organiza en categorías la información.

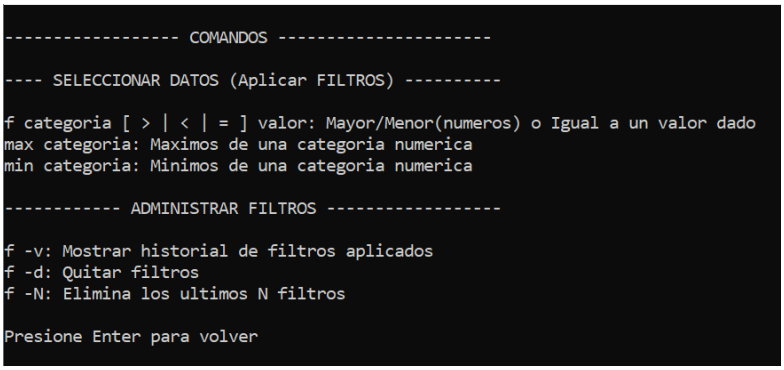

Imagen 3b. Opción 1 del menú de la imagen 2. Comandos relacionados con la selección de información a través de filtros.

El menú tiene una implementación sencilla (ante cualquier duda en el anexo del presente informe se provee el código de todo el programa), motivo por el cual no se entrará en detalle. Nos dedicaremos a abordar, únicamente, los temas centrales, suficientes y necesarios para cumplir con los objetivos antes descriptos.

Imports

Para poder abordar un codigo tan extenso y la multiplicidad de funciones provistas , fue necesario organizar la implementación en secciones o partes. Luego de la investigación pertinente (fuentes accesibles desde la bibliografía), se dividió el programa en los siguientes 7 archivos:

- py.py
- googlesheet.py
- modo_automatico.py
- filtros.py
- historial.py
- lector.py
- ejemplo_de_aplicacion.py

Variables fundamentales – Diccionarios:

La segunda decision de diseño, utilizada para afrontar la anteriormente mencionada complejidad de la implementación, fue el empleo de un Diccionario (se provee un link con la respectiva información en la Bibliografía) para almecenar el conjunto de varibles que define la escencia del programa. Dicho de otra forma, se podría: guardar esos datos en un archivo, cerrar la aplicación, volverla a abrir y por último cargar los valores nuevamente; para estar en las mismas condiciones previas a cerrar la aplicación. De hecho, de haber tenido un poco más de tiempo, se hubiera implementado dicha funcionalidad.

```
508     #El diccionario py es mi programa en si, o sea toda la informacion
509     py={"xls":xls,"hist":nuevo_historial(),"sheet":"","regs":[],"selec":[]}
510     py["auto_path"] = "./imports/modo_automatico/"
511     py["hist_cmd"] = []
512     py["delay"] = 0.2
```

Imagen 1. Diccionario que guarda la información fundamental del programa.

Google Sheet , excepciones, Time

El primer inconveniente que acarrea el vínculo con Google guarda relacion con los Errores asociados a factores externos a la aplicación , tales como: problemas de conexión, modificacion de la hoja de calculo desde el navegador (lo cual, de implementarse con los recaudos necesarios, es una gran ventaja) ó la cuota máxima de interacciones dentro de un determinado tiempo, entre otros. Por este motivo se investigó eh hizo uso (fuente en la bibliografía), por un lado, del manejo de excepciones en python; y por el otro, la funcion sleep(), para brindar mas estabilidad al programa frente a la cuota maxima antes mencionada (principalmente durante el modo automatico)

```
493 def conectarse_a_google():
494     path = "./imports/google_sheet/"
495     xls = False
496     i = "S"
497     while (not xls and i.upper() != "N"):
498         try:
499             # True = Cualquier cosa distinta de 0 o False
500             xls = cargar_xls(path,0)
501         except:
502             print("Error al cargar el archivo de Google She
503             i = input("\n¿Desea intentar de nuevo? (S/N): \
504 
505     return xls
506 
```

Imagen 5. Manejo de excepciones.
Desde el archivo py.py se llama a la función cargar_xls(), contenida en googlesheet.py.
Investigación: La condición "not xls" retorna False cuando xls tiene cualquier valor distinto de 0 o False

Con respecto a las configuraciones necesarias para obtener las credenciales y poder vincularse con Google, se deja en la bibliografía un tutorial muy claro. Si bien el entorno de Google Cloud Platform cambió (las capturas de pantalla del tutorial no condicen exactamente con la actual página web) se puede resolver de manera intuitiva ya que las funciones siguen estando vigentes, o sea, ordenadas de otra forma, pero accesibles al usuario.

```
 9  def cargar_xls(path, t):
10      json = path + "iron-cedar-331617-6743037a3a9f.json"
11      nombre_archivo = "python_goolge_sheet"
12      scope = ['https://spreadsheets.google.com/feeds','https://www.googleap
13      creds = ServiceAccountCredentials.from_json_keyfile_name(json, scope)
14      client = gspread.authorize(creds)
15      xls = client.open(nombre_archivo) #no haria falta un "close()"
16      time.sleep(t)
17      return xls
```

Imagen 6. Conexión con Google.
Archivo googlesheet.py, función que permite obtener el archivo (xls).
El código y los valores empleados guardan relación con el tutorial disponible en la bibliografía.

Interpretando comandos

En py.py se encuentran las funciones que median entre el usuario y las tareas realizadas dentro de cada archivo importado, además de las operaciones matemáticas básicas: contar elementos, total y promedio. Para ello se vale de lector.py, de modo de poder interpretar la información que ingresa el usuario, o sea, por un lado, identificar palabras y separadores tales como: >, <, = ó espacios; y por el otro, ignorar espacios o tabulaciones escritos por error por parte del usuario.

```
        while(cmd != "s"
            cmdL = cmd.lower()
517
532 #Mostrar
            elif sin_espacios_al_final(cmdL,"v")):
                imprimir(py["selec"], py)
                ...

            cmd = ""
            while(cmd==""):
591             cmd = leer_comando(py)
            cmd = cmd[espacios_al_comienzo(cmd):]

        return
```

Imagen 7. While principal en py.py. Se interpretan comandos con ayuda de lector.py
En recuadros amarillos vemos las funciones de lector.py, mientras que en verde soluciones locales de py.py

Básicamente, si algún comando es reconocido se llama a la función que termina de interpretarlo, o sea, de recuperar los parámetros presentes en el mismo. De este modo, antes de llamar a la función que realiza la tarea, en muchos casos, se optó por invocar una función intermedia, ubicada en "py.py". Fue justamente, al leer los parámetros de cada comando, que fue necesario distinguir separadores y leer palabras. Se decidió, entonces, emplear funciones que retornaran listas, para poder acceder así a múltiples productos de una misma función.

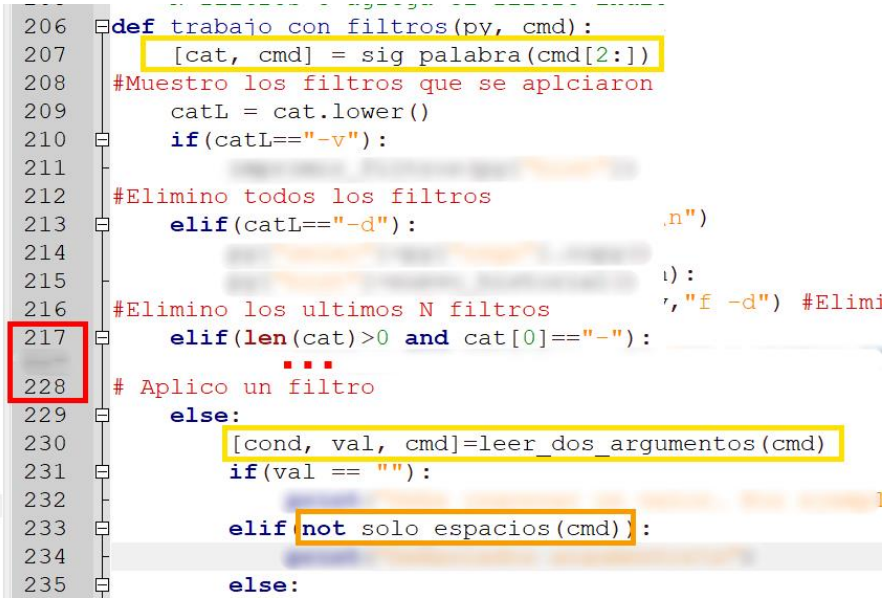

Imagen 8. Función, en py.py , que interpreta los parámetros del comando con ayuda de lector.py. En recuadros amarillos vemos las funciones para leer palabras.

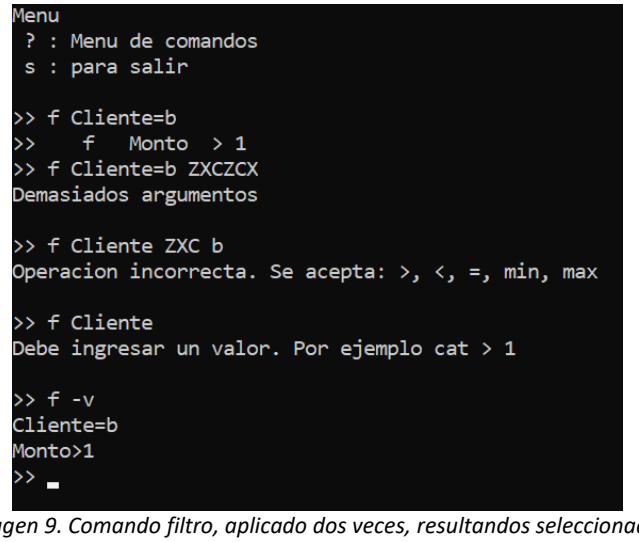

Imagen 9. Comando filtro, aplicado dos veces, resultandos seleccionados los Montos mayores a 1 realizados por el Cliente "b". El comando f -v, muestra la lista de filtros aplicados efectivamente.

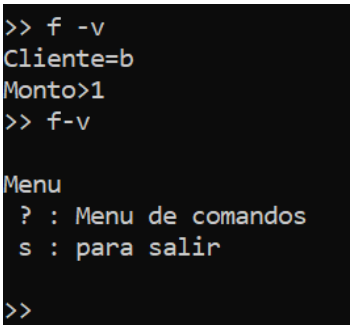

Imagen 10. Es obligatorio el espacio entre el comando y los parámetros que empiezan con guion.

Instalacion del ejemplo_de_aplicacion.py

El ejemplo de aplicación es un archivo por separado, importado en tiempo de ejecución. Consultara al programa principal , utilizando los mismos comandos descriptos anteriormente, para obtener la informacion necesaria. Pueden instalarse diferentes aplicaciones, y ejecutarse alternadamente.

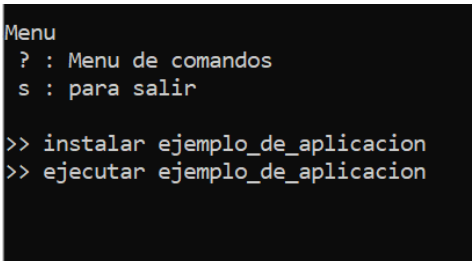

Imagen 11. Comandos para instalar y ejecutar el ejemplo de aplicación.

```
15    """ Instala una aplicacion. """
16    def install(soft):
17        soft = importlib.import_module(soft, package=None)
18        return soft.instalar()
19
```

Imagen 12. Se importa en tiempo de ejecución el módulo, o sea el archivo, cuyo nombre esta almacenado en el parámetro soft.

```
615            py["app_activa"] = {"leer":leer_comando,"ayuda":imprimir_ayuda}
616            py["app_activa"]["ayuda"]()
617            py["apps"] = {"principal":py["app_activa"]}
618            py["run_cmd"] = ejecutar_comando
```

Imagen 13. Las funciones por defecto para leer e imprimir la ayuda serán alteradas al momento de ejecutar un nuevo programa sobre el software descripto. Por otro lado, se mantiene dentro del diccionario py[“apps”] todos los programas instalados.

```
50
51    def instalar():
52        app = {"leer": leer, "ayuda":menu, "ejecutar":ejecutar}
53        return app
54
```

Imagen 14. Archivo ejemplo_de_aplicacion.py. Vemos como se crea el nuevo diccionario que será retornado al instalar el programa.

```
584     elif(cmdL[:9]=="instalar "):
585         [soft, cmd] = sig_palabra(cmdL[9:])
586         if (solo_espacios(cmd)):
587             try:
588                 py["apps"][soft]=install(soft)
589             except:
590                 print("No se pudo instalar. Tenga en cuenta que el prog
591 
592     elif(cmdL[:9]=="ejecutar "):
593         [soft, cmd] = sig_palabra(cmdL[9:])
594         if (solo_espacios(cmd) and py["apps"].get(soft) is not None):
595             [leer, ayuda, ejecutar] = py["apps"][soft]
596             py["app_activa"] = py["apps"][soft]
597             py["app_activa"]["ejecutar"](py)
598             py["app_activa"] = py["apps"]["principal"]
599         else:
600             py["app_activa"]["ayuda"]()
```

Imagen 15. La lectura de los comandos dentro del bucle principal del software, para instalar y luego ejecutar el programa. Al salir del programa se reestablecen los valores por defecto en la línea 598

ejemplo_de_aplicacion.py

Estas implementaciones que trabajaran con el software desarrollado , pueden, como en el ejemplo, tener su propio while principal. Enviaran los comandos al software desarrollado para obtener resultados con suma facilidad.

```
16 
17 def ejecutar(py):
18     os.system("cls")
19     py["run_cmd"]("h Instituto",py)
20     opt=""
21     while (opt != "8"):
22         py["run_cmd"]("f -d",py)
23         if (opt == "1"):
24             py["run_cmd"]("i",py)
25         elif (opt == "2"):
26             py["run_cmd"]("f Nota>6.5",py)
27             py["run_cmd"]("v",py)
28         elif (opt == "3"):
29             py["run_cmd"]("f Nota>6.5",py)
30             py["run_cmd"]("c",py)
31         elif (opt == "4"):
32             materia = input("Materia: ")
33             py["run_cmd"]("f Materia="+materia,py)
34             py["run_cmd"]("v",py)
35         elif (opt == "5"):
36             alumno = input("Materia: ")
37             py["run_cmd"]("f Alumno="+alumno,py)
38             if(len(py["selec"])>0):
39                 print("Si, esta matriculado en al menos una materia")
40             else:
41                 print("No, no tiene ninguna materia asignada")
42         elif (opt == "6"):
43             py["run_cmd"]("max Nota",py)
44             py["run_cmd"]("v",py)
45         elif (opt == "7"):
46             py["run_cmd"]("p Nota",py)
47         else:
48             menu()
49         opt = input("> ")
50 
```

Imagen 15. Bucle principal en ejemplo_de_aplicacion.py

Código

En los anexos 1 y 2 se presenta el codigo del programa, haciendo incapié en las funciones principales, o sea aquellas que se vinculan directamente con el enunciado del trabajo práctico.

Conclusión

Partiendo de un enunciado flexible y una base previa en programación; la cual si bien es sólida, hacia mucho tiempo que no escribia código con este nivel de complejidad; se obtuvo un producto final que: por un lado voy a utilizar en mis emprendimientos; y por el otro, me presentó una serie de retos que me pusieron a prueba en reiteradas ocaciones, brindandome así una seguridad y confianza que antes no tenía.

Las espectativas eran altas y la satisfacción final también lo es. Al mismo tiempo, el trabajo evidenció la cantidad de conocimientos que aún me faltan adquirir. Si bien se trabajó con muchos temas de investigación, cada uno de ellos merece un capítulo aparte y debo profundizar en detalle.

Por último, hay que comentar una cuestión interesante, y es el hecho de que este software es muy útil para implementar con facilidad programas que tienen las características de aquellos detallados en el enunciado. Sin embargo, es necesario estudiar otras problemáticas, de naturaleza diferente, para poder ampliar el repertorio de opciones.

ANEXO 1

El Anexo 1 consta de las funciones principales, o sea, donde se encuentran los algoritmos encargados de realizar las operaciones establecidas en el enunciado del tp. El resto del código está disponible en el Anexo 2, incluyendo por su puesto lo explicado anteriormente.

Mostrar Valores

```
24  """ Imprime los registros de un modo amigable para el usuario. Si esta
25      habilitado el modo_automatico guarda en un archivo los datos. """
26  def imprimir(registros, py):
27      if( len(registros) == 0 ):
28          print("No hay datos seleccionados")
29          #print("Filtros aplicados: "+str(len_hist(py["hist"]))+"\n")
30          return
31
32      for fila in registros:
33          values="| "
34          for k in fila.keys():
35              values = values + k + ": " + str(fila[k]) + " | "
36          print(values)
37          if(py.get("auto_cmd") is not None):
38              escribir_salidas(values, py["auto_path"])
39      return
```

Operaciones matemáticas

```
181 def tot_prom(py, cmd, operacion):
182     [cat, cmd]=sig_palabra(cmd)
183     if ( not solo_espacios(cmd) ):
184         py["app_activa"]["ayuda"]()
185         return
186     try:
187         check_categoria(py["selec"], cat)
188         operacion_txt = "Total"
189         rta = sumar(py, cat)
190         if(rta==-1):
191             print("Hay al menos un elemeno que no se puede sumar\n")
192         else:
193             if(operacion.lower()=="p"):
194                 rta = rta / len(py["selec"])
195                 operacion_txt = "Promedio"
196             print(operacion_txt+" de "+cat+": "+str(rta))
197
198     except ValueError as txt:
199         print(txt)
200     return py
```

```python
""" Suma todos los valores de la seleccion actual (o sea, los datos
    perteneciente a la actual hoja de calculo que pasaron todos los
    filtros) para una categoria dada. Si no contiene valores numericos
    retorna -1. """
def sumar(py, categoria):
    s = 0
    for e in py["selec"]:
        try: #para chequear errores, en este caso si no es un numero
            num = float(e[categoria])
            s = s + num
        except: #si no es numero, viene aca
            return -1
    return s
```

```
Menu
 ? : Menu de comandos
 s : para salir

>> v
| Monto: 1 | Cliente: a | Fecha: 1 | Producto: a | Fila: 2 |
| Monto: 2 | Cliente: b | Fecha: 2 | Producto: b | Fila: 3 |
>> p Monto
Promedio de Monto: 1.5
>> t Monto
Total de Monto: 3.0
```

Filtros, registros

La tabla de la hoja de cálculo esta compuesta por: colomnas, que se modelan en python como keys de un diccionario; y por las filas, siendo cada una un diccionario. Por lo tanto, la tabla completa (única por cada hoja) es una lista de diccionarios. A cada diccionario le llamamos registro y una selección es un subconjunto de registros obtenido despues de aplicar un filtro, ya sea sobre la totalidad de los registros, ó, sobre una selección previa.

Para evitar repetir código se hizo uso del pasaje de funciones por parámetro (investigación en bibliografía). De este modo, por ejemplo, la función filtrar puede retornar todos los registros mayores, menores o iguales a un valor aplicando un mismo procedimiento. Lo único que cambia es la función en cargada de comparar.

```python
""" Retorna una lista con los regitros (diccionarios) en los cuales
    el valor de la columna "categoria" y el parametro "valor"
    cumplen con la funcion "condicion": igual(), menor() o mayor().

    Da error si la seleccion esta vacia, la categoria es invalida, si la
    condicion esta mal (distinto de: <, >, =) o si hay en dicha categoria
    al menos un dato no numerico para las condiciones mayor y menor. """
def filtrar(categoria, condicion, valor, seleccion):
    check_categoria(seleccion, categoria)
    condicion = traducir_condicion(condicion)
    filtro = []
    for r in seleccion :
        try:
            if(condicion == igual):
                valor = str(valor)
                r_val = str(r[categoria])
            else:
                valor = float(valor)
                r_val = float(r[categoria])
        except:
            raise ValueError("Error de tipos, para > y < deberian ser sol
        if(condicion(r_val,valor)):
            filtro.append(r)
    return filtro
```

```python
def menor(a,b):
    return a<b
def igual(a,b):
    return a==b
def mayor(a,b):
    return a>b

```

```python
""" Retorna una lista con los registros (filas ,almecenadas en el
    parametro seleccion) que contienen el maximo (cuando la condicion es
    la funcion mayor()) o el minimo (cuando la condicion es la fucion
    menor()) de la categoria indicada.

    Da error si la seleccion esta vacia, la categoria es invalida , si
    la condicion esta mal (distinto de: min, max) o si en dicha
    categoria se encuentra al menos un dato no numerico. """
def min_max(categoria, condicion, seleccion):
    check_categoria(seleccion, categoria)
    condicion = traducir_condicion(condicion)
    filtro = [seleccion[0]]
    float( filtro[0][categoria] ) #chequeo categoria valida
    s=1
    while( s<len(seleccion) ):
        r = seleccion[s]
        try:
            a = float( r[categoria] )
            b = float( filtro[0][categoria] )
        except:
            raise ValueError("Error de tipos, deberian ser solo numeros\n")
        if(condicion(a,b)):
            filtro = [r]
        elif(a==b):
            filtro.append(r)
        s=s+1

    return filtro

```

Ingresar y eliminar datos

En este caso, tambien se reciben funciones por parámetro. El motivo es permitir que los datos sean obtenidos de diferentes fuentes: ingresados por el usuario, leídos desde un archivo local (modo automático) o desde una aplicación similar a ejemplo_de_aplicacion.py.

Se optó por agregar una key a cada diccionario, indicando el numero de fila. Se tuvo en cuenta el orden en que los elementos son eliminados eh ingresados a la lista. De este modo, eliminando primero las filas inferiores, se ahorró actualizar los numero de filas de aquellos que estan por encima.

```python
""" Elimina de la hoja de google sheet (libro_contable)
los registros almacenados en "seleccion" y retorna la lista actualizada
de registros

Nota: Se espera que en la lista seleccion los elementos esten ordenados
segun el numero de filas menor a mayor. Es importante que esten ordenados
porque al borrar de la hoja de calculo cambian los numeros de las filas
por debajo de la que es eliminada. """
def eliminar(seleccion, libro_contable, registros, t):
    c=len(seleccion)
    if ( c>0 ):
        if(seleccion==registros):
            libro_contable.delete_rows(2,c+1) #borro todo
            time.sleep(t)
            registros=[]
        else:
            menor_fila = seleccion[0]["Fila"] #los cambios se realizan de esta fila ha
            while(c>0):
                libro_contable.delete_rows(seleccion[c-1]["Fila"])
                time.sleep(t)
                registros.remove(seleccion[c-1])#los diccionarios no son copias, son l
                c=c-1 #empiezo del final, leer descripcion de la funcion
            actualizado=enum_filas(registros[menor_fila-2:],menor_fila)#la primer fila
            registros=registros[:menor_fila-2] + actualizado
    return registros #Para cumplir con lo visto en la materia

```

```
84  """ Retorna un registro(diccionario) con la nueva entrada despues de
85      anotarla en la hoja de calculo. Retorna None en caso de no ser
86      posible completar la accion, sin modificar la hoja de calculo.
87      Los posibles problemas tienen que ver con el servidor o porque la
88      funcion lector() no pudo leer correctamente alguno de los datos.
89
90      La funcion que recibe como parametro, debe tener la siguiete
91      estructura:
92          lector(categoria, datos)
93      La misma debe retornar los valores que seran agregados a una nueva
94      fila de la hoja de google sheet (sheet).
95
96       """
97  def anotar(sheet, lector, datos, t):
98      col = 1
99      fil = len(sheet.col_values(1))+1
100     time.sleep(t)
101     categorias = sheet.row_values(1)
102     registro={}
103     sheet.add_rows(fil)#me aseguro que halla una fila
104     time.sleep(t)
105     try:
106         for k in categorias:
107             valor = lector(k, datos)
108             sheet.update_cell(fil,col,valor)
109             time.sleep(t)
110             registro[k]= valor#en programa() cumple enunciado cargando lista
111             col=col+1
112         registro["Fila"] = fil
113     except:
114         sheet.delete_rows(fil)
115         time.sleep(t)
116         raise ValueError()
117
118     return registro
119
```

ANEXO 2

Debido a la gran extension que adquirió el código (tan solo el archivo py.py tiene mas de 600 lineas) no tiene ningún sentido copiar y pegar aquí, por ello se optó por brindar los path , partiendo de la carpeta “raíz” del programa , para poder acceder a ellos con facilidad y leerlos dentro de un enterno amigable como Geany o inlcuso imprimirlos.

Los diferentes archivos que componene el programa en su totalidad estan ordenados en carpetas, su ubicación es importante, no debe ser alterada.

- py.py
- googlesheet.py
- modo_automatico.py
- filtros.py
- historial.py
- lector.py
- ejemplo_de_aplicacion.py

Bibliografía

Manejo de excepciones:
https://uniwebsidad.com/libros/algoritmos-python/capitulo-12/excepciones

Enunciado del TP Final
https://drive.google.com/file/d/1O0KXRqb8meUI789wtcQR8tcKph5SAGk5/view?usp=sharing

Tutorial para usar Google Sheets: “Read and Update Google Spreadsheets with Python!”
https://www.analyticsvidhya.com/blog/2020/07/read-and-update-google-spreadsheets-with-python/

Import: “Módulos y Librerías en Python : Importar, acceder, crear“
https://pythones.net/importar-modulos-en-python/

Time: “Python sleep()“
https://www.programiz.com/python-programming/time/sleep

“Diccionarios en Python “
https://devcode.la/tutoriales/diccionarios-en-python/

Enviar una función como parámetro: “Passing function as an argument in Python”
https://www.geeksforgeeks.org/passing-function-as-an-argument-in-python/

importar un módulo cuando el nombre del módulo está en una variable
https://stackoverflow.com/questions/13598035/importing-a-module-when-the-module-name-is-in-a-variable